



CMSC 5743

Efficient Computing of Deep Neural Networks

Mo03: Quantization

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Latest update: February 20, 2026)

2026 Spring



These slides contain/adapt materials developed by

- Hardware for Machine Learning, Shao Spring 2020 @ UCB
- 8-bit Inference with TensorRT
- **Amir Gholami et al. (2021). “A survey of quantization methods for efficient neural network inference”. In: *arXiv preprint***



- ① Floating Point Number
- ② Integer & Fixed-Point Number
- ③ Quantization Overview
- ④ Quantization – First Example
- ⑤ Post Training Quantization (PTQ)
- ⑥ Quantization Aware Training (QAT)



Floating Point Number



Scientific notation: 6.6254×10^{-27}

- A normalized number of certain accuracy (e.g. 6.6254 is called the **mantissa**)
- Scale factors to determine the position of the decimal point (e.g. 10^{-27} indicates position of decimal point and is called the exponent; the **base** is implied)
- **Sign** bit

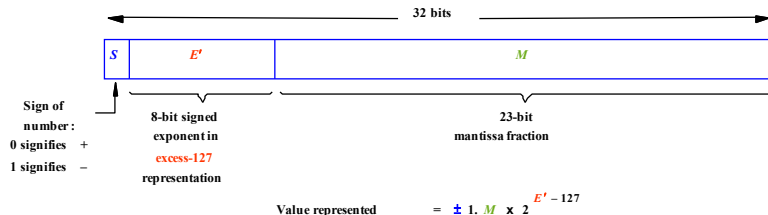


- Floating Point Numbers can have multiple forms, e.g.

$$\begin{aligned}0.232 \times 10^4 &= 2.32 \times 10^3 \\&= 23.2 \times 10^2 \\&= 2320. \times 10^0 \\&= 232000. \times 10^{-2}\end{aligned}$$

- It is desirable for each number to have a unique representation => **Normalized Form**
- We normalize Mantissa's in the Range $[1..R)$, where R is the Base, e.g.:
 - $[1..2)$ for BINARY
 - $[1..10)$ for DECIMAL

32-bit, float in C / C++ / Java



(a) Single precision



$$\text{Value represented} = +1.001010 \dots 0 \times 2^{-87}$$

(b) Example of a single-precision number

00101000 → 40

$$40 - 127 = -87$$

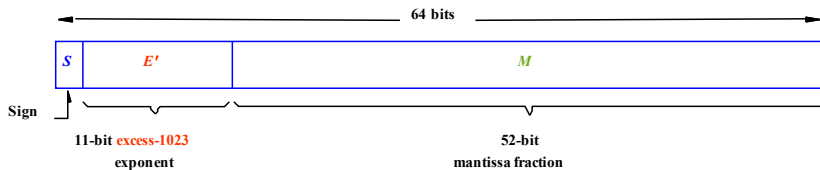


Note:

- minimum exponent: $1 - 127 = -126$
- maximum exponent: $254 - 127 = 127$
- Why 254? If exponents are all 1, the floating num has special values (please refer to following part)



64-bit, float in C / C++ / Java



$$\text{Value represented} = \pm 1. M \times 2^{E' - 1023}$$

(c) Double precision



Question:

What is the IEEE single precision number $40C0\ 0000_{16}$ in decimal?



Question:

What is the IEEE single precision number $40C0\ 0000_{16}$ in decimal?

- Binary: $0100\ 0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000$
- Sign: $+$
- Exponent: $129 - 127 = +2$
- Mantissa: $1.100\ 0000\ \dots_2 \rightarrow 1.5_{10} \times 2^{+2}$
- $\rightarrow +110.0000\ \dots_2$
- Decimal Answer = $+6.0_{10}$



Question:

What is -0.5_{10} in IEEE single precision binary floating point format?



Question:

What is -0.5_{10} in IEEE single precision binary floating point format?

- Binary: $-1.0... \times 2^{-1}$ (in binary)
- Exponent: $127 + (-1) = 01111110$
- Sign bit: 1
- Mantissa: 1.000 0000 0000 0000 0000 0000
- Binary representation: 1011 1111 0000 0000 0000 0000 0000 0000



Exponents of all 0's and all 1's have special meaning

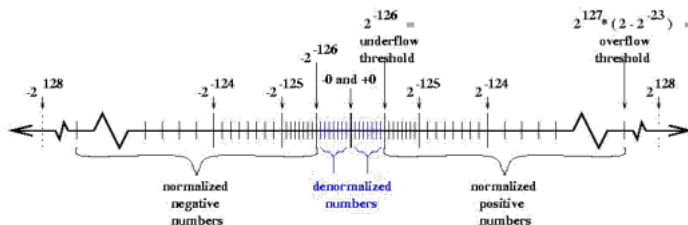
- $E=0, M=0$ represents 0 (sign bit still used so there is ± 0)
- $E=0, M \neq 0$ is a denormalized number $\pm 0.M \times 2^{-126}$ (smaller than the smallest normalized number)
- $E=\text{All } 1\text{'s}, M=0$ represents $\pm \text{Infinity}$, depending on Sign
- $E=\text{All } 1\text{'s}, M \neq 0$ represents NaN

- Normalized $\pm 1.d\dots d \times 2^{\text{exp}}$
- **Denormalized** $\pm 0.d\dots d \times 2^{\text{min_exp}}$ $\rightarrow$ to represent *near-zero* numbers
e.g. $+ 0.0000\dots 0000001 \times 2^{-126}$ for Single Precision

Format	# bits	# significant bits	macheps	# exponent bits	exponent range
Single	32	1+23	2^{-24} ($\sim 10^{-7}$)	8	$2^{-126} - 2^{+127}$ ($\sim 10^{\pm 38}$)
Double	64	1+52	2^{-53} ($\sim 10^{-16}$)	11	$2^{-1022} - 2^{+1023}$ ($\sim 10^{\pm 308}$)
Double Extended	≥ 80	≥ 64	$\leq 2^{-64}$ ($\sim 10^{-19}$)	≥ 15	$2^{-16382} - 2^{+16383}$ ($\sim 10^{\pm 4932}$)

(Double Extended is 80 bits on all Intel machines)
macheps = Machine Epsilon = $2 - (\# \text{ significand bits})$

$$\epsilon_{\text{mach}}$$





Note:

- Smallest normalized: $1.000\ 0000\ \dots\ 0000_2 \times 2^{-126} = 2^{-126}$
- Largest denormalized: $0.111\ 1111\ \dots\ 1111_2 \times 2^{-126} = (1 - 2^{-1/23}) \times 2^{-126}$
- Smallest denormalized: $0.000\ 0000\ \dots\ 0001_2 \times 2^{-126} = 2^{-149}$
- Smallest denormalized value is much closer to 0



Example: Find 1st root of a quadratic equation¹

$$r = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Expected: 0.00023025562642476431

Double: 0.00023025562638524986

Float: 0.00024670246057212353

¹On Sparc processor, Solaris, gcc 3.3 (ANSI C)



Example: Find 1st root of a quadratic equation¹

$$r = \frac{-b + \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Expected: 0.00023025562642476431

Double: 0.00023025562638524986

Float: 0.00024670246057212353

- Problem is that if c is near zero, $\sqrt{b^2 - 4 \cdot a \cdot c} \approx b$
- **Rule of thumb:** use the highest precision which does not give up too much speed

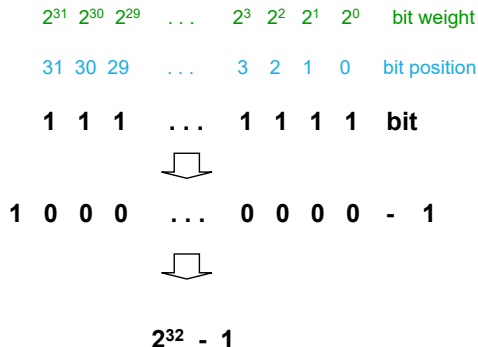
¹On Sparc processor, Solaris, gcc 3.3 (ANSI C)



Integer & Fixed-Point Number



Hex	Binary	Decimal
0x00000000	0...0000	0
0x00000001	0...0001	1
0x00000002	0...0010	2
0x00000003	0...0011	3
0x00000004	0...0100	4
0x00000005	0...0101	5
0x00000006	0...0110	6
0x00000007	0...0111	7
0x00000008	0...1000	8
0x00000009	0...1001	9
	...	
0xFFFFFFFFC	1...1100	$2^{32} - 4$
0xFFFFFFFFD	1...1101	$2^{32} - 3$
0xFFFFFFFFE	1...1110	$2^{32} - 2$
0xFFFFFFFFF	1...1111	$2^{32} - 1$



Signed Binary Representation



	2'sc binary	decimal
$-2^3 =$	1000	-8
$-(2^3 - 1) =$	1001	-7
	1010	-6
	1011	-5
	1100	-4
	1101	-3
	1110	-2
	1111	-1
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7

complement all the bits

0101 1011

and add a 1 and add a 1

0110 1010

complement all the bits

$2^3 - 1 =$



- Integers with a binary point and a bias
 - “slope and bias”: $y = s \cdot x + z$
 - Qm.n: m (# of integer bits) n (# of fractional bits)

$s = 1, z = 0$

2^2	2^1	2^0	Val
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

$s = 1/4, z = 0$

2^0	2^{-1}	2^{-2}	Val
0	0	0	0
0	0	1	1/4
0	1	0	2/4
0	1	1	3/4
1	0	0	1
1	0	1	5/4
1	1	0	6/4
1	1	1	7/4

$s = 4, z = 0$

2^4	2^3	2^2	Val
0	0	0	0
0	0	1	4
0	1	0	8
0	1	1	12
1	0	0	16
1	0	1	20
1	1	0	24
1	1	1	28

$s = 1.5, z = 10$

2^2	2^1	2^0	Val
0	0	0	$1.5 \cdot 0 + 10$
0	0	1	$1.5 \cdot 1 + 10$
0	1	0	$1.5 \cdot 2 + 10$
0	1	1	$1.5 \cdot 3 + 10$
1	0	0	$1.5 \cdot 4 + 10$
1	0	1	$1.5 \cdot 5 + 10$
1	1	0	$1.5 \cdot 6 + 10$
1	1	1	$1.5 \cdot 7 + 10$



$(a - b)$ is **inaccurate** when $a \gg b$ or $a \ll b$

Decimal Example 1:

- Using **2 significant digits**
- Compute mean of 5.1 and 5.2 using the formula $(a + b)/2$:
- $a + b = 10$ (with 2 significant digits, 10.3 can only be stored as 10)
- $10/2 = 5.0$ (the computed mean is less than both numbers!!!)

Decimal Example 2:

- Using **8 significant digits** to compute sum of three numbers:
- $(11111113 + (-11111111)) + 7.5111111 = 9.5111111$
- $11111113 + ((-11111111) + 7.5111111) = 10.000000$

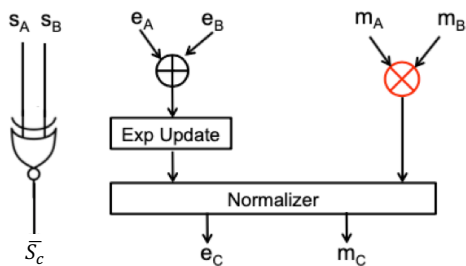


Catastrophic cancellation occurs when

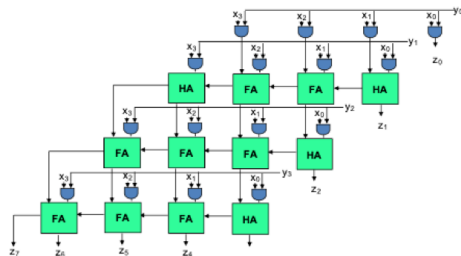
$$\left| \frac{[\text{round}(x) \times \text{round}(y)] - \text{round}(x \times y)}{\text{round}(x \times y)} \right| \gg \epsilon$$

Multipliers

Multiplier Example: $C = A \times B$



Floating-point multiplier

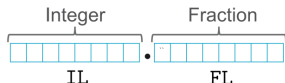


Fixed-point multiplier



Fixed-Point Arithmetic

Number representation $\langle \text{IL}, \text{FL} \rangle$



Word Length $\text{WL} = \text{IL} + \text{FL}$

Granularity $2^{-\text{FL}}$

Range $[-2^{\text{IL}-1}, 2^{\text{IL}-1} - 2^{-\text{FL}}]$

Convert $(x, \langle \text{IL}, \text{FL} \rangle) =$

$$\begin{cases} -2^{\text{IL}-1} & \text{if } x \leq -2^{\text{IL}-1} \\ 2^{\text{IL}-1} - 2^{-\text{FL}} & \text{if } x \geq 2^{\text{IL}-1} - 2^{-\text{FL}} \\ \text{Round}(x, \langle \text{IL}, \text{FL} \rangle) & \text{otherwise} \end{cases}$$



Fixed-Point Arithmetic

Number representation $\langle \text{IL}, \text{FL} \rangle$



Word Length $\text{WL} = \text{IL} + \text{FL}$

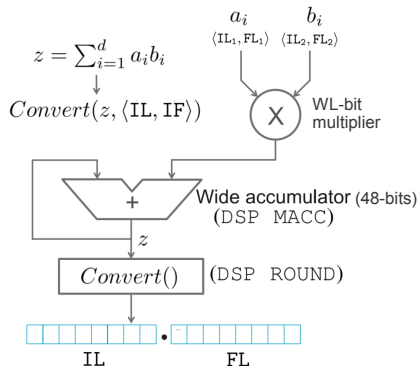
Granularity $2^{-\text{FL}}$

Range $[-2^{\text{IL}-1}, 2^{\text{IL}-1} - 2^{-\text{FL}}]$

$\text{Convert}(x, \langle \text{IL}, \text{FL} \rangle) =$

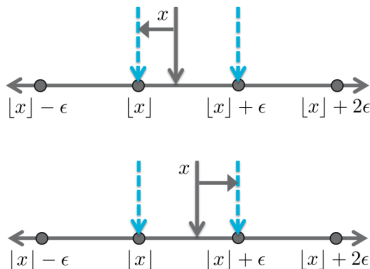
$$\begin{cases} -2^{\text{IL}-1} & \text{if } x \leq -2^{\text{IL}-1} \\ 2^{\text{IL}-1} - 2^{-\text{FL}} & \text{if } x \geq 2^{\text{IL}-1} - 2^{-\text{FL}} \\ \text{Round}(x, \langle \text{IL}, \text{FL} \rangle) & \text{otherwise} \end{cases}$$

Multiply-and-ACCumulate



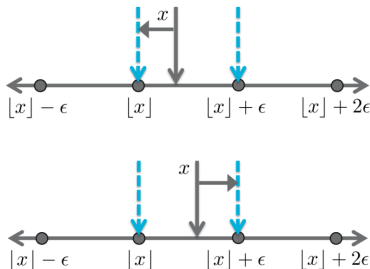
Fixed-Point Arithmetic: Rounding Modes

Round-to-nearest

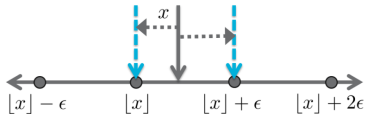


Fixed-Point Arithmetic: Rounding Modes

Round-to-nearest



Stochastic rounding



$\text{Round}(x, \langle \text{IL}, \text{FL} \rangle) =$

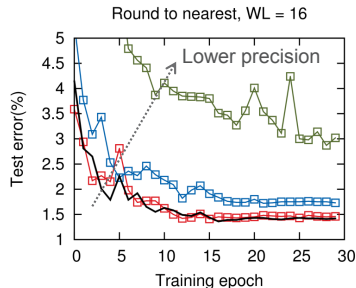
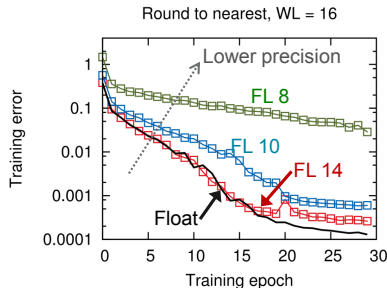
$$\begin{cases} \lfloor x \rfloor & \text{w.p. } 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & \text{w.p. } \frac{x - \lfloor x \rfloor}{\epsilon} \end{cases}$$

- Non-zero probability of rounding to either $\lfloor x \rfloor$ or $\lfloor x \rfloor + \epsilon$
- Unbiased rounding scheme: expected rounding error is zero



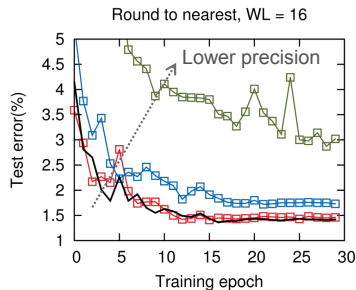
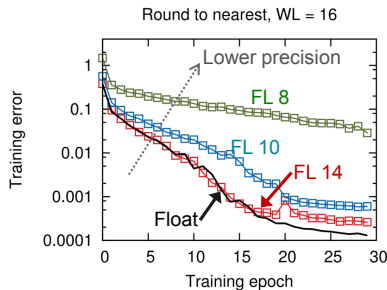


MNIST: *Fully-connected DNNs*





MNIST: *Fully-connected DNNs*

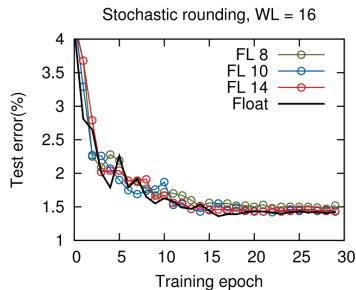
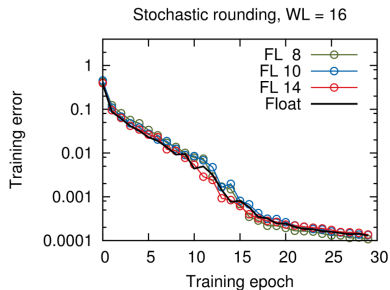


- For small fractional lengths (FL < 12), a large majority of weight updates are rounded to zero when using the round-to-nearest scheme.
 - Convergence slows down
- For FL < 12, there is a noticeable degradation in the classification accuracy





MNIST: *Fully-connected DNNs*



- Stochastic rounding preserves gradient information (statistically)
 - No degradation in convergence properties
- Test error nearly equal to that obtained using 32-bit floats



The diagram illustrates the high-level architecture of the proposed hardware accelerator, implemented on a Xilinx Kintex K325T FPGA. The architecture is divided into three main functional blocks: Memory, Control, and Computation.

Memory: The system features 8GB DDR3 SO-DIMM memory. This memory is connected to an AXI Interface to DDR3, which is linked to a READ/WRITE block and an L2 Cache (BRAM). The L2 Cache is connected to the Systolic Array (SA) via an L2-to-SA block.

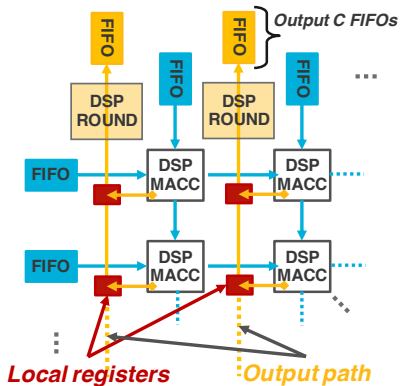
Control: The Top Controller manages the system. It is connected to the AXI Interface to DDR3 and the Systolic Array (SA) of Multiple-and-Accumulate (MACC) units.

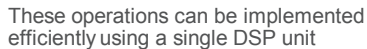
Computation: The Systolic Array (SA) is an $n \times n$ array of DSP MACC units. It receives input data from Input FIFOs: Matrix A and Input FIFOs: Matrix B. The SA is connected to the L2 Cache via an L2-to-SA block. The SA is also connected to the L2 Cache via an L2-to-SA block.

The diagram also shows the communication path between the L2 Cache and the SA, labeled "Communication".

Wavefront systolic array for computing matrix product $\mathbf{AB}$. Arrows indicate dataflow

Stochastic rounding







Quantization Overview

Quantization:

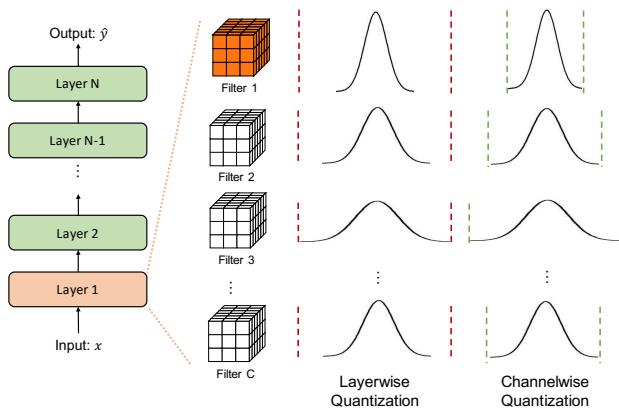
$$Q(r) = \text{Int}(r/S) - Z$$

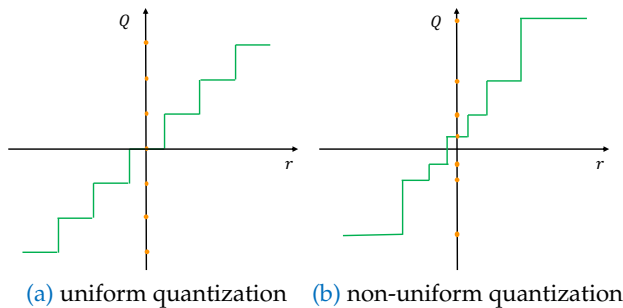
Dequantization:

$$\hat{r} = S(Q(r) + Z)$$

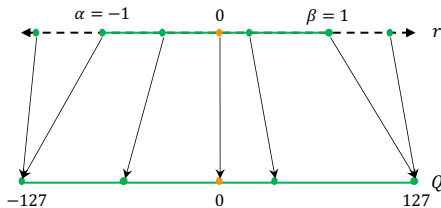
Granularity:

- Layerwise
- Groupwise
- Channelwise

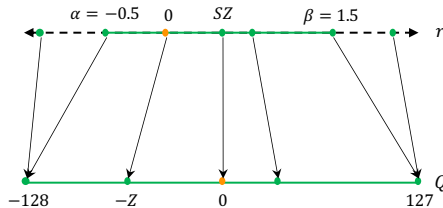




- Real values in the continuous domain r are mapped into discrete
- Lower precision values in the quantized domain Q .
- **Uniform** quantization: distances between quantized values are **the same**
- **Non-uniform** quantization: distances between quantized values can **vary**

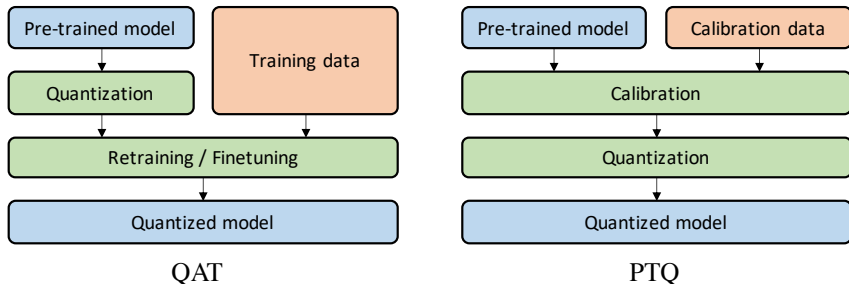


(a) Symmetric quantization



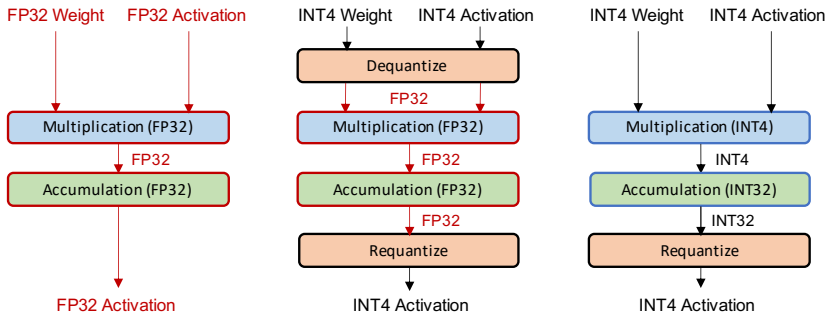
(b) Asymmetric quantization

- Symmetric vs. Asymmetric: $Z = 0$?
- Fig. (a) Symmetric w. **restricted range** maps $[-127, 127]$,
- Fig. (b) Asymmetric w. **full range** maps to $[-128, 127]$
- Both for 8-bit quantization case.



- **quantization-aware training (QAT)**: model is quantized using training data to adjust parameters and recover accuracy degradation.
- **post-training quantization (PTQ)**: a pre-trained model is calibrated using finetuning data (e.g., a small subset of training data) to compute the clipping ranges and the scaling factors.
- **Key difference**: Model parameters fixed/unfixed.

Simulated quantization vs Integer-Only quantization



Left : Full-precision

Middle : Simulated quantization

Right : Integer-only quantization



Hardware Support

- Nvidia GPU: Tensor Core support FP16, Int8 and Int4
- Arm: Neon 128-bit SIMD instruction: $4 \times 32\text{bit}$ or $8 \times 16\text{bit}$ up to $16 \times 8\text{bit}$
- Intel: SSE intrinsics, same as above
- DSP, AI Chip

Some common architectures:

- For CPU: Tensorflow Lite, QNNPACK, NCNN
- For GPU: TensorRT
- Versatile Compiler such TVM.qnn



Quantization – First Example



Linear quantization

Representation:

Tensor Values = FP32 scale factor * int8 array + FP32 bias



Do we really need bias?

Two matrices:

$$A = \text{scale_A} * QA + \text{bias_A}$$

$$B = \text{scale_B} * QB + \text{bias_B}$$

Let's multiply those 2 matrices:

$$\begin{aligned} A * B = & \text{scale_A} * \text{scale_B} * QA * QB + \\ & \text{scale_A} * QA * \text{bias_B} + \\ & \text{scale_B} * QB * \text{bias_A} + \\ & \text{bias_A} * \text{bias_B} \end{aligned}$$



Do we really need bias?

Two matrices:

$$A = \text{scale_A} * QA + \text{bias_A}$$

$$B = \text{scale_B} * QB + \text{bias_B}$$

Let's multiply those 2 matrices:

$$\begin{aligned} A * B &= \text{scale_A} * \text{scale_B} * QA * QB + \\ &\quad \cancel{\text{scale_A} * QA * \text{bias_B}} \neq \\ &\quad \cancel{\text{scale_B} * QB * \text{bias_A}} \neq \\ &\quad \cancel{\text{bias_A} * \text{bias_B}} \end{aligned}$$



Do we really need bias? No!

Two matrices:

$$A = \text{scale_A} * QA$$

$$B = \text{scale_B} * QB$$

Let's multiply those 2 matrices:

$$A * B = \text{scale_A} * \text{scale_B} * QA * QB$$



Symmetric linear quantization

Representation:

Tensor Values = FP32 scale factor * int8 array

One FP32 scale factor for the entire int8 tensor

Q: How do we set scale factor?



MINIMUM QUANTIZED VALUE

- Integer range is not completely symmetric. E.g. in 8bit, [-128, 127]
 - If use [-127, 127], $s = \frac{127}{\alpha}$
 - Range is symmetric
 - 1/256 of int8 range is not used. 1/16 of int4 range is not used
 - If use full range [-128, 127], $s = \frac{128}{\alpha}$
 - Values should be quantized to 128 will be clipped to 127
 - Asymmetric range may introduce bias



EXAMPLE OF QUANTIZATION BIAS

Bias introduced when int values are in $[-128, 127]$

$$A = [-2.2 \quad -1.1 \quad 1.1 \quad 2.2], B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix}, AB = 0$$

8bit scale quantization, use $[-128, 127]$. $s_A = 128/2.2$, $s_B = 128/0.5$

$$[-128 \quad -64 \quad 64 \quad 127] * \begin{bmatrix} 127 \\ 77 \\ 77 \\ 127 \end{bmatrix} = -127$$

Dequantize -127 will get -0.00853 . A small bias is introduced towards $-\infty$



EXAMPLE OF QUANTIZATION BIAS

No bias when int values are in $[-127, 127]$

$$A = [-2.2 \quad -1.1 \quad 1.1 \quad 2.2], B = \begin{bmatrix} 0.5 \\ 0.3 \\ 0.3 \\ 0.5 \end{bmatrix}, AB = 0$$

8-bit scale quantization, use $[-127, 127]$. $s_A=127/2.2$, $s_B=127/0.5$

$$[-127 \quad -64 \quad 64 \quad 127] * \begin{bmatrix} 127 \\ 76 \\ 76 \\ 127 \end{bmatrix} = 0$$

Dequantize 0 will get 0



MATRIX MULTIPLY EXAMPLE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$



MATRIX MULTIPLY EXAMPLE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range (scale $127/2=63.5$) for first matrix and [-1, 1] fp range (scale = $127/1=127$) for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$



MATRIX MULTIPLY EXAMPLE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range (scale $127/2=63.5$) for first matrix and [-1, 1] fp range (scale = $127/1=127$) for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

The result has an overall scale of $63.5 * 127$. We can *dequantize* back to float

$$\begin{pmatrix} -5222 \\ -3413 \end{pmatrix} * \frac{1}{63.5 * 127} = \begin{pmatrix} -0.648 \\ -0.423 \end{pmatrix}$$



REQUANTIZE

Scale Quantization

$$\begin{pmatrix} -1.54 & 0.22 \\ -0.26 & 0.65 \end{pmatrix} * \begin{pmatrix} 0.35 \\ -0.51 \end{pmatrix} = \begin{pmatrix} -0.651 \\ -0.423 \end{pmatrix}$$

8bit quantization

choose [-2, 2] fp range for first matrix and [-1, 1] fp range for the second

$$\begin{pmatrix} -98 & 14 \\ -17 & 41 \end{pmatrix} * \begin{pmatrix} 44 \\ -65 \end{pmatrix} = \begin{pmatrix} -5222 \\ -3413 \end{pmatrix}$$

Requantize output to a different quantized representation with fp range [-3, 3]:

$$\begin{pmatrix} -5222 \\ -3413 \end{pmatrix} * \frac{127/3}{63.5 * 127} = \begin{pmatrix} -27 \\ -18 \end{pmatrix}$$



Post Training Quantization (PTQ)



- For a fixed-point number, its representation is:

$$n = \sum_{i=0}^{bw-1} B_i \cdot 2^{-f_l} \cdot 2^i,$$

where bw is the bit width and f_l is the fractional length which is dynamic for different layers and feature map sets while static in one layer.

- Weight quantization: find the optimal f_l for weights:

$$f_l = \arg \min_{f_l} \sum |W_{float} - W(bw, f_l)|,$$

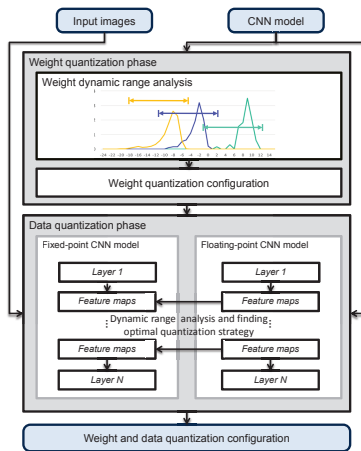
where W is a weight and $W(bw, f_l)$ represents the fixed-point format of W under the given bw and f_l .

³Jiantao Qiu et al. (2016). “Going deeper with embedded fpga platform for convolutional neural network”. In: *Proc. FPGA*, pp. 26–35.

- Feature quantization: find the optimal f_l for features:

$$f_l = \arg \min_{f_l} \sum |x_{float}^+ - x^+(bw, f_l)|,$$

where x^+ represents the result of a layer when we denote the computation of a layer as $x^+ = A \cdot x$.

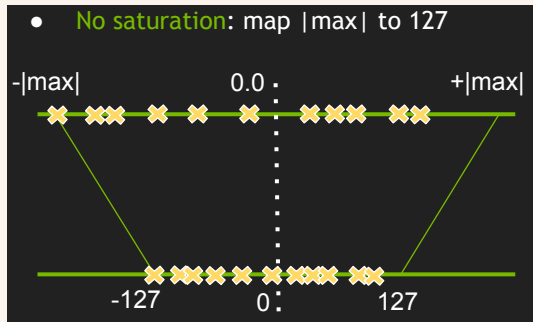




Network	VGG16						
Data Bits	Single-float	16	16	8	8	8	8
Weight Bits	Single-float	16	8	8	8	8	8 or 4
Data Precision	N/A	2^{-2}	2^{-2}	Impossible	$2^{-5}/2^{-1}$	Dynamic	Dynamic
Weight Precision	N/A	2^{-15}	2^{-7}	Impossible	2^{-7}	Dynamic	Dynamic
Top-1 Accuracy	68.1%	68.0%	53.0%	Impossible	28.2%	66.6%	67.0%
Top-5 Accuracy	88.0%	87.9%	76.6%	Impossible	49.7%	87.4%	87.6%

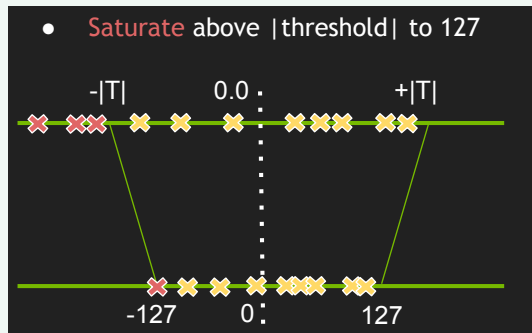
Network	CaffeNet			VGG16-SVD		
Data Bits	Single-float	16	8	Single-float	16	8
Weight Bits	Single-float	16	8	Single-float	16	8 or 4
Data Precision	N/A	Dynamic	Dynamic	N/A	Dynamic	Dynamic
Weight Precision	N/A	Dynamic	Dynamic	N/A	Dynamic	Dynamic
Top-1 Accuracy	53.9%	53.9%	53.0%	68.0%	64.6%	64.1%
Top-5 Accuracy	77.7%	77.1%	76.6%	88.0%	86.7%	86.3%

No Saturation Quantization – INT8 Inference



- Map the maximum value to 127, with uniform step length.
- Suffer from outliers.

Saturation Quantization – INT8 Inference



- Set a threshold as the maximum value.
- Divide the value domain into 2048 groups.
- Traverse all the possible thresholds to find the best one with minimum KL divergence.



Relative Entropy of two encodings

- INT8 model encodes the same information as the original FP32 model.
- Minimize the loss of information.
- Loss of information is measured by **Kullback-Leibler divergence** (*a.k.a.*, relative entropy or information divergence).
 - P, Q - two discrete probability distributions:

$$D_{KL}(P\|Q) = \sum_{i=1}^N P(x_i) \log \frac{P(x_i)}{Q(x_i)}$$

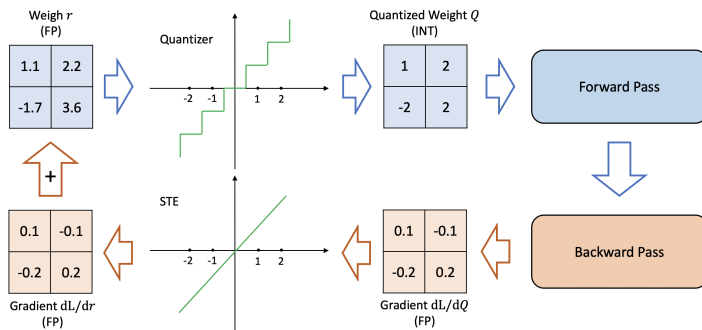
- Intuition: KL divergence measures **the amount of information lost** when approximating a given encoding.



Quantization Aware Training (QAT)

Straight Through Estimator (STE)⁴

- Forward integer, Backward floating point
- Rounding to nearest



⁴Yoshua Bengio, Nicholas Léonard, and Aaron Courville (2013). “Estimating or propagating gradients through stochastic neurons for conditional computation”. In: *arXiv preprint arXiv:1308.3432*.



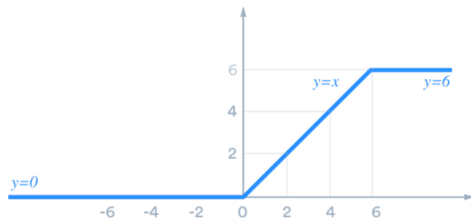
Is Straight-Through Estimator (STE) the best?

- Gradient mismatch: the gradients of the weights are not generated using the value of weights, but rather its quantized value.
- Poor gradient: STE fails at investigating better gradients for quantization training.

PArameterized Clipping acTivation (PACT)⁵

- Relu6 $\rightarrow$ clipping
- threshold $\rightarrow$ clipping range in quantization
- range upper/lower bound trainable

$$y = PACT(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$



⁵Jungwook Choi, Zhuo Wang, et al. (2018). "Pact: Parameterized clipping activation for quantized neural networks". In: *arXiv preprint arXiv:1805.06085*.



- A new activation quantization scheme in which the activation function has a parameterized clipping level α .
- The clipping level is dynamically adjusted via stochastic gradient descent (SGD)-based training with the goal of minimizing the quantization error.
- In PACT, the convolutional ReLU activation function in CNN is replaced with:

$$f(x) = 0.5 (|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in [0, \alpha) \\ \alpha, & x \in [\alpha, +\infty) \end{cases}$$

where α limits the dynamic range of activation to $[0, \alpha]$.

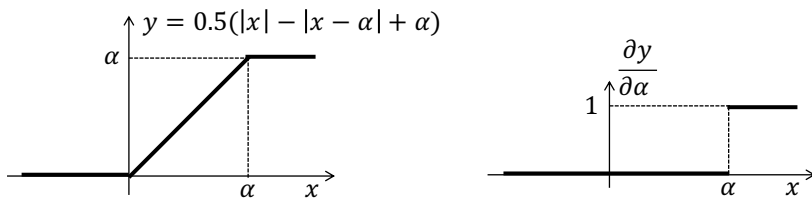
⁶Jungwook Choi, Swagath Venkataramani, et al. (2019). “Accurate and efficient 2-bit quantized neural networks”. In: *Proceedings of Machine Learning and Systems* 1.



- The truncated activation output is the linearly quantized to k -bits for the dot-product computations:

$$y_q = \text{round} \left(y \cdot \frac{2^k - 1}{\alpha} \right) \cdot \frac{\alpha}{2^k - 1}$$

- With this new activation function, α is a variable in the loss function, whose value can be optimized during training.
- For back-propagation, gradient $\frac{\partial y_q}{\partial \alpha}$ can be computed using STE to estimate $\frac{\partial y_q}{\partial y}$ as 1.



PACT activation function and its gradient.